AN2352
Application note

PLL jitter effects on C-CAN modules of the ST10F27x

Introduction

PLL (Phase Locked Loop) is increasingly used in microcontrollers to achieve higher internal clock frequencies. This improves performance while reducing overall noise. One drawback in the use of PLL circuits is that they create a small but still measurable level of transient phase shifts, or *jitter*. The aim of this note is to describe the effects of PLL jitter on the C-CAN modules present in the devices of the family ST10F27x.

This document begins with a brief guide on configuring the bit time of the CAN protocol and then goes on to cover the characteristics of the PLL as implemented in the ST10F27x. The last section shows the results of the effect of the ST10F27x PLL on the C-CAN.

The information contained in this document is valid for the ST10F27x, ST10R27x, ST10F25x and ST10F296 devices.

Contents

1 Configuration of the CAN bit timing 3

1.1 Bit time and bit rate 3

1.2 Propagation time segment 4

1.3 Phase buffer segments and synchronization 5

1.4 System clock tolerance range 8

1.5 Configuration of the CAN protocol controller 10

1.6 Calculation of the bit timing parameters 12

1.6.1 Example for bit timing at high baud rate 13

1.6.2 Example for bit timing at low baud rate 14

2 Phase locked loop (PLL) 15

2.1 PLL jitter 15

2.1.1 Jitter in the input clock 15

2.1.2 Noise in the PLL loop 15

3 ST10F27x PLL jitter effect in CAN protocol 17

3.1 System clock tolerance range reduction in presence of PLL jitter 17

3.1.1 Range reduction percentage 17

3.1.2 Examples 18

3.2 Conclusion 18

4 Revision history 19

1 Configuration of the CAN bit timing

Even if minor errors in the configuration of the CAN bit timing do not result in immediate failure, the performance of a CAN network can be reduced significantly.

In many cases, the CAN bit synchronization compensates a faulty configuration of the CAN bit timing to such a degree that an error frame is generated only occasionally. In the case of arbitration however, when two or more CAN nodes simultaneously try to transmit a frame, a misplaced sample point may cause one of the transmitters to become error passive.

The analysis of such sporadic errors requires a detailed knowledge of the CAN bit synchronization inside a CAN node and of the CAN nodes' interaction on the CAN bus.

1.1 Bit time and bit rate

CAN supports bit rates in the range of less than 1 Kbit/s up to 1000 Kbit/s. Each member of the CAN network has its own clock generator, usually a quartz oscillator. The timing parameter of the bit time (that is, the reciprocal of the bit rate) can be configured individually for each CAN node, creating a common bit rate even though the CAN nodes' oscillator periods (f_{osc}) may be different.

The frequencies of these oscillators are not absolutely stable. Small variations are caused by changes in temperature or voltage and by deteriorating components. As long as the variations remain within a specific oscillator tolerance range (df), the CAN nodes can compensate the different bit rates by resynchronizing to the bit stream.

According to the CAN specifications, the bit time is divided into four segments (*Figure 1*):

- Synchronization segment
- Propagation time segment
- Phase buffer segment 1
- Phase buffer segment 2

Each segment consists of a specific, programmable number of time quanta (see *Table 1*). The length of the time quantum (t_q),which is the basic time unit of the bit time, is defined by the CAN controller's system clock f_{sys} and the Baud Rate Prescaler (BRP):

$$t_q = BRP / f_{sys}$$

The C-CAN's system clock f_{sys} is the f_{CPU} or f_{CPU} / 2 according to bit 2 of the XMSIC register.

The synchronization segment (Sync_Seg) is that part of the bit time where edges of the CAN bus level are expected to occur; the distance between the Sync_Seg and an edge that occurs outside of Sync_Seg is called the phase error of that edge. The propagation time segment (Prop_Seg) is intended to compensate the physical delay times within the CAN network. The phase buffer segments (Phase_Seg1 and Phase_Seg2) surround the sample point. The (Re)synchronization jump width (SJW) defines how far a resynchronization may move the sample point within the limits defined by the phase buffer segments to compensate edge phase errors.

Figure 1. Bit timing

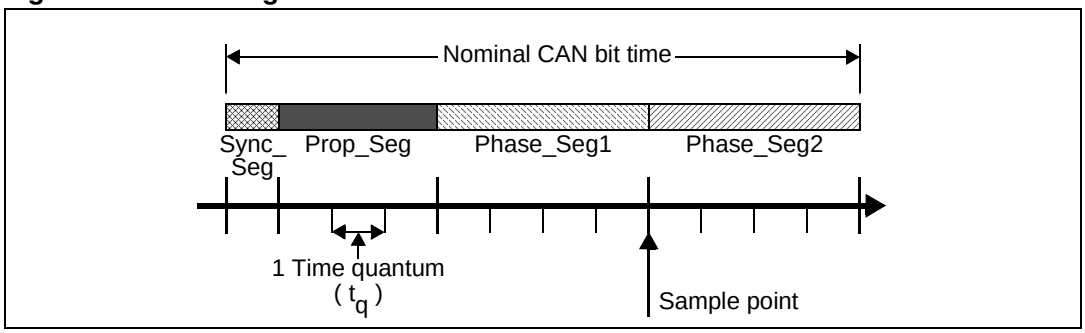


Table 1 describes the minimum programmable ranges required by the CAN protocol.

Table 1. Parameters of the CAN bit time

Parameter	Range	Remark
BRP	[1 .. 32]	Defines the length of the time quantum t_q
Sync_Seg	1 t_q	Fixed length, synchronization of bus input to system clock
Prop_Seg	[1 .. 8] t_q	Compensates the physical delay times
Phase_Seg1	[1 .. 8] t_q	May be lengthened temporarily by synchronization
Phase_Seg2	[1 .. 8] t_q	May be shortened temporarily by synchronization
SJW	[1 .. 4] t_q	May not be longer than either phase buffer segment

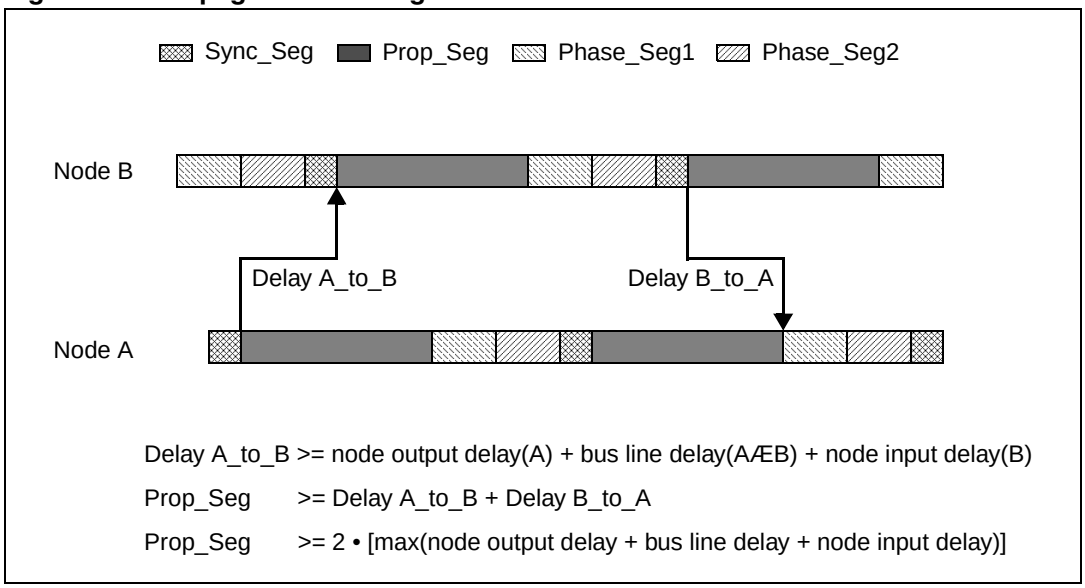
A given bit rate may be met by different bit time configurations but for the proper functioning of the CAN network, the physical delay times and the oscillator's tolerance range must be taken into consideration.

1.2 Propagation time segment

This part of the bit time compensates physical delay times within the network. These delay times consist of the signal propagation time on the bus and the internal delay time of the CAN nodes.

Any CAN node synchronized to the bit stream on the CAN bus will be out of phase with the transmitter of that bit stream due to the signal propagation time between the two nodes. The CAN protocol's nondestructive bit-wise arbitration and the dominant acknowledge bit provided by the receivers of CAN messages require a CAN node transmitting a bit stream to also receive dominant bits transmitted by other CAN nodes that are synchronized to that bit stream. The example in *Figure 2* shows the phase shift and propagation times between two CAN nodes.

Figure 2. Propagation time segment


Sync_Seg
Prop_Seg
Phase_Seg1
Phase_Seg2
Node B
Delay A_to_B
Delay B_to_A
Node A
Delay A_to_B >= node output delay(A) + bus line delay(AÆB) + node input delay(B)
Prop_Seg >= Delay A_to_B + Delay B_to_A
Prop_Seg >= 2 • [max(node output delay + bus line delay + node input delay)]

In this example, both nodes A and B are transmitters performing an arbitration for the CAN bus. Node A has sent its start of frame bit less than one bit time earlier than node B, therefore node B has synchronized itself to the received edge from recessive to dominant. Since node B has received this edge delay (A_to_B) after it has been transmitted, B's bit timing segments are shifted in relation to A. Node B sends an identifier with a higher priority and therefore will win the arbitration at a specific identifier bit when it transmits a dominant bit while node A transmits a recessive bit. The dominant bit transmitted by node B arrives at node A after the delay (B_to_A).

Due to oscillator tolerances, the actual position of node A's sample point can be anywhere inside the nominal range of node A's phase buffer segments, so the bit transmitted by node B must arrive at node A before the start of Phase_Seg1. This condition defines the length of Prop_Seg.

If the edge from recessive to dominant transmitted by node B would arrive at node A after the start of Phase_Seg1, it is possible that node A samples a recessive bit instead of a dominant bit, resulting in a bit error and the destruction of the current frame by an error flag.

The error occurs only when two nodes with oscillators at opposite ends of the tolerance range and separated by a long bus line arbitrate for the CAN bus; this is an example of a minor error in the bit timing configuration (Prop_Seg too short) that causes sporadic bus errors.

1.3 Phase buffer segments and synchronization

The phase buffer segments (Phase_Seg1 and Phase_Seg2) and the synchronization jump width (SJW) are used to compensate the oscillator tolerance. The phase buffer segments may be lengthened or shortened by synchronization.

Synchronizations occur on edges from recessive to dominant; their purpose is to control the distance between edges and sample points.

Edges are detected by sampling the actual bus level in each time quantum and comparing it with the bus level at the previous sample point. A synchronization is possible only if a

recessive bit was sampled at the previous sample point and if the actual time quantum's bus level is dominant.

An edge is synchronous if it occurs inside of Sync_Seg, otherwise the distance between an edge and the end of Sync_Seg is the edge phase error, measured in time quanta. If the edge occurs before Sync_Seg, the phase error is negative, otherwise, it is positive.

Two types of synchronization exist: hard synchronization and resynchronization. A hard synchronization occurs once at the start of a frame. Resynchronizations occur only inside a frame.

- **Hard synchronization**

  After a hard synchronization, the bit time is restarted at the end of Sync_Seg, regardless of the edge phase error. Thus hard synchronization forces the edge which placed the hard synchronization into the synchronization segment of the restarted bit time.

- **Bit resynchronization**

  Resynchronization leads to a shortening or lengthening of the bit time such that the position of the sample point is shifted in relation to the edge.

  When a positive phase error of the edge causes resynchronization, Phase_Seg1 is lengthened. If the magnitude of the phase error is less than the SJW, Phase_Seg1 is lengthened by the magnitude of the phase error, otherwise it is lengthened by the SJW.

  When a negative phase error of the edge causes resynchronization, Phase_Seg2 is shortened. If the magnitude of the phase error is less than SJW, Phase_Seg2 is shortened by the magnitude of the phase error, otherwise it is shortened by the SJW.

When the magnitude of the phase error of the edge is less than or equal to the programmed value of the SJW, the results of hard synchronization and resynchronization are the same. If the magnitude of the phase error is greater than the SJW, the resynchronization cannot entirely compensate the phase error and an error of (phase error - SJW) remains.

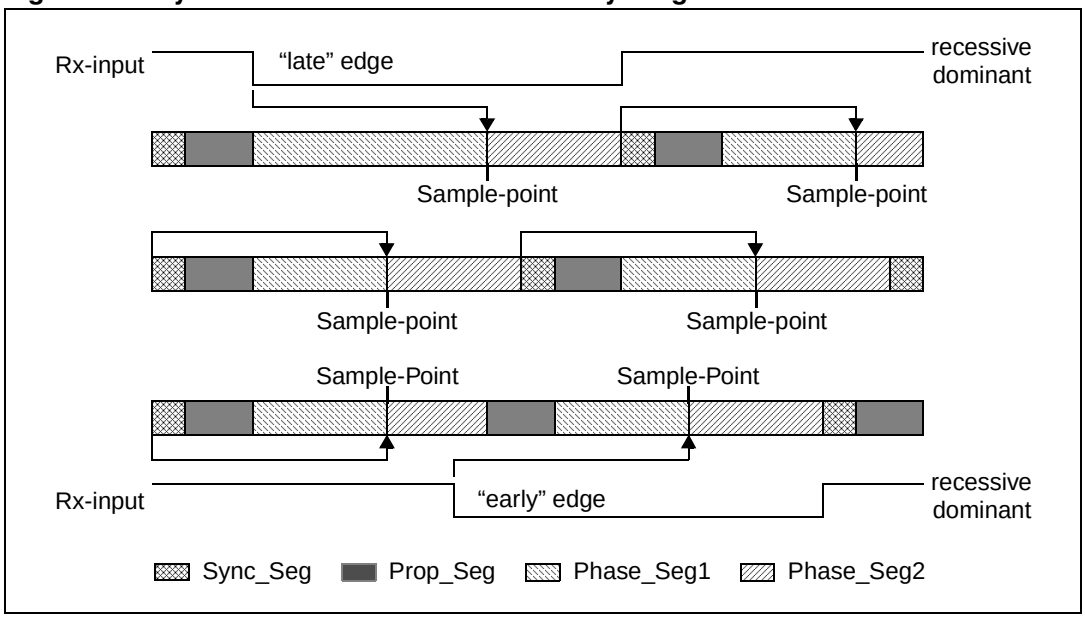
Only one synchronization occurs between two sample points. The synchronizations maintain a minimum distance between edges and sample points, giving the bus level time to stabilize and filtering out spikes that are shorter than (Prop_Seg + Phase_Seg1).

Apart from noise spikes, most synchronizations are caused by arbitration. All nodes synchronize "hard" on the edge transmitted by the "leading" transceiver that started transmitting first, but due to propagation delay times, they are not ideally synchronized. The "leading" transmitter does not necessarily win the arbitration, therefore the receivers must synchronize themselves to different transmitters that subsequently "take the lead" and that are synchronized differently to the previously "leading" transmitter. The same happens at the acknowledge field, where the transmitter and some of the receivers must synchronize to the receiver that "takes the lead" in the transmission of the dominant acknowledge bit.

Synchronizations after the end of the arbitration are caused by oscillator tolerance, when the differences in the oscillator's clock periods of transmitter and receivers sum up during the time between synchronizations (maximum 10 bits). These summarized differences may not be longer than the SJW, limiting the oscillator's tolerance range.

The examples in *Figure 3* show how the phase buffer segments compensate phase errors. There are three drawings of each two consecutive bit timings. The upper drawing shows the synchronization on a "late" edge, the lower drawing shows the synchronization on an "early" edge and the middle drawing is the reference without synchronization.

Figure 3. Synchronization on “late” and “early” edges



In the first example, an edge from recessive to dominant occurs at the end of Prop_Seg. The edge is “late” since it occurs after the Sync_Seg. Reacting to the “late” edge, Phase_Seg1 is lengthened so that the distance from the edge to the sample point is the same as it would have been from the Sync_Seg to the sample point if no edge had occurred. The phase error of this “late” edge is less than the SJW, so it is fully compensated and the edge from dominant to recessive at the end of the bit, which is one nominal bit time long, occurs in the Sync_Seg.

In the second example, an edge from recessive to dominant occurs during Phase_Seg2. The edge is “early” since it occurs before a Sync_Seg. Reacting to the “early” edge, Phase_Seg2 is shortened and Sync_Seg is omitted, so that the distance from the edge to the sample point is the same as it would have been from an Sync_Seg to the sample point if no edge had occurred. As in the previous example, the magnitude of this “early” edge’s phase error is less than the SJW, so it is fully compensated.

The phase buffer segments are lengthened or shortened only temporarily; at the next bit time, the segments return to their nominal programmed values.

In these examples, the bit timing is seen from the point of view of the CAN implementation’s state machine, where the bit time starts and ends at the sample points. The state machine omits Sync_Seg when synchronizing on an “early” edge because it cannot subsequently redefine that time quantum of Phase_Seg2 where the edge occurs to be the Sync_Seg.

1.4 System clock tolerance range

The CAN system clock for the different nodes in the network is typically derived from a different clock generator source. The actual CAN system clock frequency for each node (and consequently the actual bit time) is affected by a tolerance. In particular, for the ST10F27x, the system clock is derived (prescaled) from the CPU clock, typically generated by the main oscillator's on-chip PLL multiplying frequency.

For effective communication, all CAN nodes in the network must sample the correct value for each transmitted bit. Also, those nodes (typically at opposite ends of the network) with the largest propagation delay and working with system clocks that are at opposite limits of the frequency tolerance, must correctly receive and decode every message transmitted on the network.

Considering the effect of the system clock discrepancy between two CAN nodes and supposing no bus errors are detected (due to, for instance, electrical disturbances), bit stuffing guarantees that even in the worst case condition for the accumulation of phase errors (during normal communication), the maximum time between two resynchronization edges is 10 bit periods (5 dominant bits followed by 5 recessive bits are always followed by a dominant bit).

Calling t_{BT} the CAN Bit Time, this maximum time t_J between two resynchronization edges can be simply expressed as:

$$t_J = 10 \cdot t_{BT}$$

Then, assuming the two CAN nodes have opposite system clock generator tolerance (considering the specified tolerance "df" valid for both nodes in the network) for their respective system clocks, the accumulated phase error at the resynchronization instant becomes:

$$\Delta t_J = (2 \cdot df) \cdot 10 \cdot t_{BT}$$

where "df" represents the system clock relative tolerance (f actual frequency, f_N nominal frequency):

$$df = \frac{|f - f_N|}{f_N}$$

This error must be compensated, therefore it must be less than the programmed (Re)Synchronization Jump Width (SJW). Calling t_{SJW} the duration of the resynchronization segment (programmable from 1 to 4 time quanta), the following condition can be written:

$$(2 \cdot df) \cdot 10 \cdot t_{BT} < t_{SJW}$$

This expression can be seen as a condition for the CAN system clock tolerance df:

$$df < \frac{(t_{SJW})}{2 \cdot 10 \cdot t_{BT}}$$

Considering now that real systems typically operate in the presence of electrical disturbances, errors on the CAN bus may occur. When an error is detected, an error flag is transmitted on the bus: If the error is just local, only the node which detected it transmits the error flag on the bus, while the other nodes simply receive the error flag and then transmit their own error flags as an echo. On the contrary, if the error is global, all nodes detect it within the same bit time and transmit their own error flags simultaneously. In this way, each

node recognizes if the error is local or global by simply detecting that there is an echo after its error flag. This is possible only if the node can properly sample the first bit after transmitting the error flag.

The error flag from an error active node is composed of 6 dominant bits; in the worst case condition of a bit stuffing error, up to 6 other dominant bits could be received before the error flag. This means that the first bit after the error flag is the 13th bit after the last synchronization: This bit, as already stated, must be correctly sampled.

Again, t_{BT} being the CAN bit time, the maximum time t_S (with correct sampling) between two resynchronization edges can be expressed as:

$$t_S = 13 \cdot t_{BT} - t_{PB2}$$

where t_{PB2} corresponds to the duration of Phase_Seg2 (PB = phase buffer).

Also in this case, assuming the two CAN nodes have opposite system clock generator tolerance (considering the specified tolerance "df" valid for both nodes in the network) for their respective system clocks, the accumulated phase error at the resynchronization instant becomes:

$$\Delta t_S = (2 \cdot df) \cdot (13 \cdot t_{BT} - t_{PB2})$$

For a correct sampling, the accumulated phase error must not lead the resynchronization edge outside the interval Phase_Seg1 + Phase_Seg2. This condition can be expressed as:

$$t_{PB1} < (2 \cdot df) \cdot (13 \cdot t_{BT} - t_{Seg2}) < t_{PB2}$$

Once again, this expression can be translated into a condition for the CAN system clock tolerance df:

$$df < \frac{\min(t_{PB1}, t_{PB2})}{(2 \cdot (13 \cdot t_{BT} - t_{PB2}))}$$

In conclusion, both conditions on the CAN system clock tolerance must be satisfied.

In case the CAN node generates its system clock through a PLL, the maximum allowed clock tolerance must also be a function of the PLL jitter**:** This results in a more severe quality requirement for the oscillator (crystal or resonator).

The phase error introduced by the PLL jitter is linked to the number of clock periods: In particular, the jitter increases with the clock period number until a saturation maximum value is attained**,** which results in the long term jitter (refer to datasheet for more details about the PLL electrical characteristics).

Considering the PLL effect, the two expressions producing the phase error in the two conditions above are modified as follows:

$$\Delta t_J = 2 \cdot (df \cdot 10 \cdot t_{BT} - \delta_{PLL})$$

$$\Delta t_S = 2 \cdot [df \cdot (13 \cdot t_{BT} - t_{PB2}) + \delta_{PLL}]$$

where δ_{PLL} represents the absolute deviation introduced by the PLL jitter. In the two formulas, the value of δ_{PLL} is evaluated for different numbers of clock periods: For the first, the jitter corresponding to a 10-bit time period must be considered, while for the second, the jitter corresponding to a 13-bit time period must be considered. The number of clock periods

is computed taking into account the baud rate prescaler setting as well. Again, the factor 2, which multiplies the single CAN node phase deviation, takes into account the worst case eventuality that the two communicating nodes are at the opposite limits of the specified frequency tolerance.

From the two equations above, the new constraints for the CAN system clock tolerance can be translated into new quality requirements for the oscillator:

$$df < \frac{(t_{SJW}) - (2 \cdot \delta_{PLL})}{2 \cdot 10 \cdot t_{BT}}$$

$$df < \frac{\min(t_{PB1}, t_{PB2}) - (2 \cdot \delta_{PLL})}{(2 \cdot (13 \cdot t_{BT} - t_{PB2}))}$$

It is evident that PLL jitter imposes more stringent constraints on oscillator tolerance than what is acceptable when no PLL is used. ST10F27x PLL characteristics are such that the oscillator requirements are acceptably impacted by jitter for the majority of the worst case CAN bus network configurations, as discussed in detail in *Section 3: ST10F27x PLL jitter effect in CAN protocol*.

It must be considered that the SJW may not be larger than the smaller of the phase buffer segments and that the propagation time segment limits the part of the bit time that may be used for the phase buffer segments.

The combination Prop_Seg = 1 and Phase_Seg1 = Phase_Seg2 = SJW = 4 allows the largest possible frequency tolerance of 1.58% (in the absence of PLL jitter). This combination with a propagation time segment of only 10% of the bit time is not suitable for short bit times; it can be used for bit rates of up to 125 Kbit/s (bit time = 8µs) with a bus length of 40m.

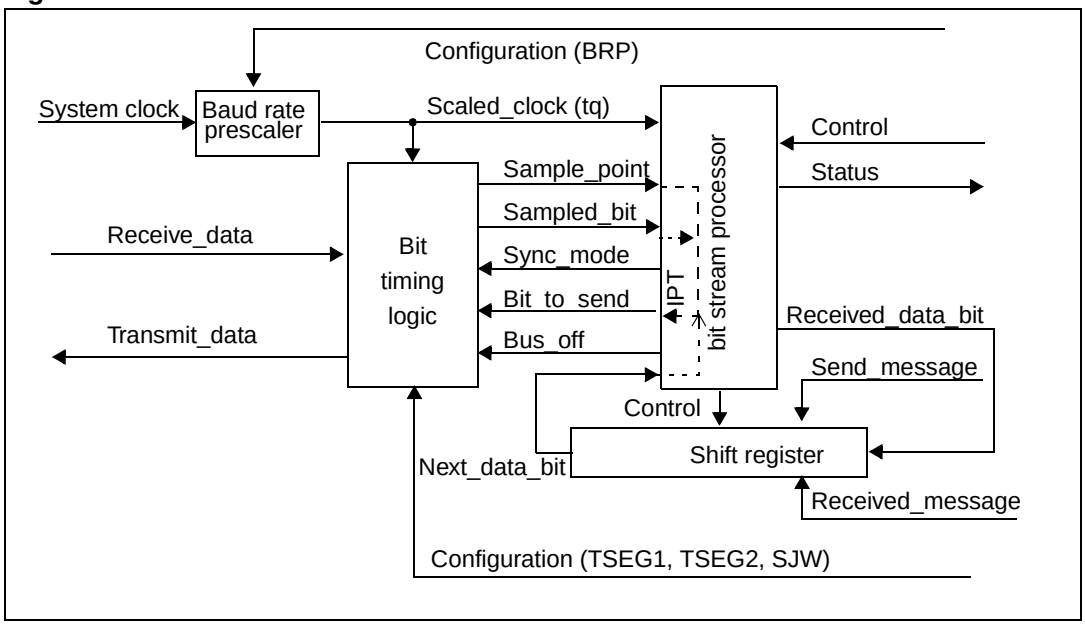
1.5 Configuration of the CAN protocol controller

In most CAN implementations and also in the C-CAN, the bit timing configuration is programmed in 2 register bytes. The sum of Prop_Seg and Phase_Seg1 (as TSEG1) is combined with Phase_Seg2 (as TSEG2) in 1 byte; SJW and BRP are combined in the other byte.

In these bit timing registers, the four components TSEG1, TSEG2, SJW and BRP must be programmed to a numerical value that is one less than its functional value; values in the range of [0...n-1] are programmed instead of values in the range of [1...n]. That way, for example, SJW (functional range of [1...4]) is represented by only 2 bits.

Therefore, the length of the bit time is (programmed values) [TSEG1 + TSEG2 + 3] t_q or (functional values) [Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_Seg2] t_q.

Figure 4. Structure of the CAN Core's CAN Protocol Controller



The data in the bit timing registers is the configuration input of the CAN protocol controller. The Baud rate prescaler (whose value is configured by the BRP field of the bit timing register) defines the length of the time quantum, the basic time unit of the bit time; the bit timing logic (configured by TSEG1, TSEG2 and SJW) defines the number of time quanta in the bit time.

The processing of the bit time, the calculation of the position of the sample point and occasional synchronizations are controlled by the BTL state machine, which is evaluated once each time quantum. The rest of the CAN protocol controller, the bit stream processor (BSP) state machine, is evaluated once each bit time at the sample point.

The Shift Register serializes the messages to be sent and parallelizes received messages. Its loading and shifting is controlled by the BSP.

The BSP translates messages into frames and frames into messages. It generates and discards the enclosed fixed format bits, inserts and extracts stuff bits, calculates and checks the CRC code, performs the error management and determines the type of synchronization to be used. It is evaluated at the sample point and processes the sampled bus input bit. The time needed to calculate the next bit to be sent after the sample point (for example, data bit, CRC bit, stuff bit, error flag or idle) is called the information processing time (IPT).

The IPT is application specific but may not be longer than 2 t_q; the C-CAN's IPT is 0 t_q. Its length is the lower limit of the programmed length of Phase_Seg2. In case of a synchronization, Phase_Seg2 may be shortened to a value less than IPT, which does not affect bus timing.

1.6 Calculation of the bit timing parameters

Usually, the calculation of the bit timing configuration starts with a desired bit rate or bit time. The resulting bit time (1/bit rate) must be an integer multiple of the system clock period.

The bit time consists of 4 to 25 time quanta. The length of the time quantum t_q is defined by the Baud rate prescaler with t_q = (Baud rate prescaler)/f_{sys}. Several combinations may lead to the desired bit time, allowing iterations of the following steps.

The first part of the bit time to be defined is the Prop_Seg. Its length depends on the delay times measured in the system. A maximum bus length as well as a maximum node delay must be defined for expandable CAN bus systems. The resulting time for Prop_Seg is converted into time quanta (rounded up to the nearest integer multiple of t_q).

The Sync_Seg is 1 t_q long (fixed), leaving (bit time – Prop_Seg – 1) t_q for the two phase buffer segments. If the number of remaining t_q is even, the phase buffer segments have the same length, Phase_Seg2 = Phase_Seg1, otherwise Phase_Seg2 = Phase_Seg1 + 1.

The minimum nominal length of Phase_Seg2 must also be considered. Phase_Seg2 may not be shorter than the CAN controller's information processing time, which, depending on the actual implementation, is in the range of [0...2] t_q.

The length of the synchronization jump width is set to its maximum value, which is the minimum of 4 and Phase_Seg1.

The oscillator tolerance range necessary for the resulting configuration is calculated by the formulas expressed in *Section 1.4: System clock tolerance range*.

If more than one configuration is possible, that configuration allowing the highest oscillator tolerance range should be chosen.

CAN nodes with different system clocks require different configurations to attain the same bit rate. The calculation of the propagation time in the CAN network, based on the nodes with the longest delay times, is done once for the whole network.

The CAN system's oscillator tolerance range is limited by that node with the lowest tolerance range.

The calculation may show that bus length or bit rate must be decreased or that the oscillator frequencies' stability must be increased in order to find a protocol compliant configuration of the CAN bit timing.

The resulting configuration is written into the bit timing register:

(Phase_Seg2 - 1) and (Phase_Seg1 + Prop_Seg - 1) and (SynchronizationJumpWidth - 1) and (Prescaler - 1)

1.6.1 Example for bit timing at high baud rate

In this example, CPU clock frequency is 40 MHz, the frequency of the CAN module clock is 20 MHz, BRP is 1 and the bit rate is 1 Mbit/s. By default, the ST10F27x divides by 2 the clock that feeds the CAN cells (mandatory when the CPU clock frequency is higher than 40 MHz). It is possible to disable this function by setting bit 2 of the XMISC register.

t_q	100 ns	$= 2 \times t_{CAN\_CLK}$
delay of bus driver	50 ns	
delay of receiver circuit	30 ns	
delay of bus line (40m)	220 ns	
t_{Prop}	600 ns	$= 6 \times t_q$
t_{PB1}	100 ns	$= 1 \times t_q$
t_{TSeg1}	700 ns	$= t_{Prop} + t_{PB1}$
t_{TSeg2}	200 ns	= Information processing time + 1 • t_q
t_{SJW}	100 ns	$= 1 \times t_q$
$t_{Sync\text{-}Seg}$	100 ns	$= 1 \times t_q$
bit time	1000 ns	$= t_{Sync\text{-}Seg} + t_{TSeg1} + t_{TSeg2}$
tolerance for CAN clock		=

$$\min\left(\frac{\min(PB1, PB2)}{2 \cdot (13 \cdot bit\_time - PB2)}, \frac{(t_{SJW})}{2 \cdot 10 \cdot t_{BT}}\right)$$

$$\frac{\min(PB1, PB2)}{2 \cdot (13 \cdot bit\_time - PB2)} = \frac{0.1\mu s}{2 \cdot (13 \cdot 1\mu s - 0.2\mu s)} = 0,39\%$$

$$\frac{(t_{SJW})}{2 \cdot 10 \cdot t_{BT}} = \frac{0.1\mu s}{2 \cdot (10 \cdot 1\mu s)} = 0,5\%$$

tolerance for CAN clock = 39%

In this example, the concatenated bit time parameters are $(2\text{-}1)_3$ and $(7\text{-}1)_4$ and $(1\text{-}1)_2$ and $(1)_6$ and the Bit Timing Register is programmed to = 0x1601.

1.6.2 Example for bit timing at low baud rate

In this example, CPU clock frequency is 64 MHz, the frequency of CAN module clock is 32 MHz, BRP is 31 and the bit rate is 100 Kbit/s. By default, the ST10F27x divides by 2 the clock that feeds the CAN cells (mandatory when the CPU clock frequency is higher than 40 MHz). It is possible to disable this function by setting bit 2 of the XMISC register.

t_q	1 µs	= 32 x t_{CAN_CLK}
delay of bus driver	200 ns	
delay of receiver circuit	80 ns	
delay of bus line (40m)	220 ns	
t_{Prop}	1 µs	= 1 x t_q
t_{PB1}	4 µs	= 4 x t_q
t_{TSeg1}	5 µs	= t_{Prop} + t_{PB1}
t_{TSeg2}	4 µs	= Information Processing Time + 3 • t_q
t_{SJW}	4 µs	= 4 x t_q
$t_{Sync\text{-}Seg}$	1 µs	= 1 x t_q
bit time	10 µs	= $t_{Sync\text{-}Seg}$ + t_{TSeg1} + t_{TSeg2}
tolerance for CAN clock		=

$$\min\left(\frac{\min(PB1, PB2)}{2 \cdot (13 \cdot bit\_time - PB2)}, \frac{(t_{SJW})}{2 \cdot 10 \cdot t_{BT}}\right)$$

$$\frac{\min(PB1, PB2)}{2 \cdot (13 \cdot bit\_time - PB2)} = \frac{4\mu s}{2 \cdot (13 \cdot 10\mu s - 4\mu s)} = 1.58\%$$

$$\frac{(t_{SJW})}{2 \cdot 10 \cdot t_{BT}} = \frac{4\mu s}{2 \cdot (10 \cdot 10\mu s)} = 2\%$$

tolerance for CAN clock = 1.58%

In this example, the concatenated bit time parameters are $(4\text{-}1)_3$ and $(5\text{-}1)_4$ and $(4\text{-}1)_2$ and $(31)_6$ and the bit timing register is programmed to = 0x34DF.

2 Phase locked loop (PLL)

The internal operation of the ST10F27x is controlled by the internal CPU clock f_{CPU}. The CPU clock signal can be generated by different mechanisms. The mechanism used to generate the CPU clock is selected during reset by the logic levels on pins P0.15-13 (P0H.7-5).

When pins P0.15-13 (P0H.7-5) equal ‘001’ during reset, the CPU clock is derived from the internal oscillator (input clock signal) by a 2:1 prescaler.

When pins P0.15-13 (P0H.7-5) equal ‘011’ during reset, the on-chip phase locked loop is disabled, the on-chip oscillator amplifier is bypassed and the CPU clock is directly driven by the input clock signal on a specific pin (XTAL1).

For all other combinations of pins P0.15-13 (P0H.7-5) during reset, the on-chip phase locked loop is enabled and it provides the CPU clock. The PLL multiplies the input frequency by the factor F which is selected via the combination of pins P0.15-13 ($f_{CPU} = f_{XTAL} \times F$). With every F’th transition of f_{cpu} the PLL circuit synchronizes the CPU clock to the input clock. This synchronization occurs smoothly, to avoid an abrupt change in the CPU clock frequency.

2.1 PLL jitter

Two kinds of PLL jitter are defined:

- Self referred single period jitter
  Also called "Period Jitter", it can be defined as the difference of the T_{max} and T_{min}, where T_{max} is the maximum time period of the PLL output clock and T_{min} is the minimum time period of the PLL output clock.
- Self referred long term jitter
  Also called "N period jitter", it can be defined as the difference of T_{max} and T_{min}, where T_{max} is the maximum time difference between N + 1 clock rising edges and T_{min} is the minimum time difference between N + 1 clock rising edges. Here N should be kept sufficiently large to have the long term jitter. For N = 1, this becomes the single period jitter.

Jitter at the PLL output is caused by:

- Jitter in the input clock
- Noise in the PLL loop

2.1.1 Jitter in the input clock

PLL acts like a low pass filter for any jitter in the input clock. Input Clock jitter with the frequencies within the PLL loop bandwidth is passed to the PLL output and higher frequency jitter (frequency > PLL bandwidth) is attenuated at 20dB/decade.

2.1.2 Noise in the PLL loop

This condition again is attributed to the following sources:

- Device noise of the circuit in the PLL
- Noise in supply and substrate

Device noise of the circuit in the PLL

Long term jitter is inversely proportional to the bandwidth of the PLL: The wider the loop bandwidth, the lower the jitter due to noise in the loop. Moreover, long term jitter is practically independent of the multiplication factor.

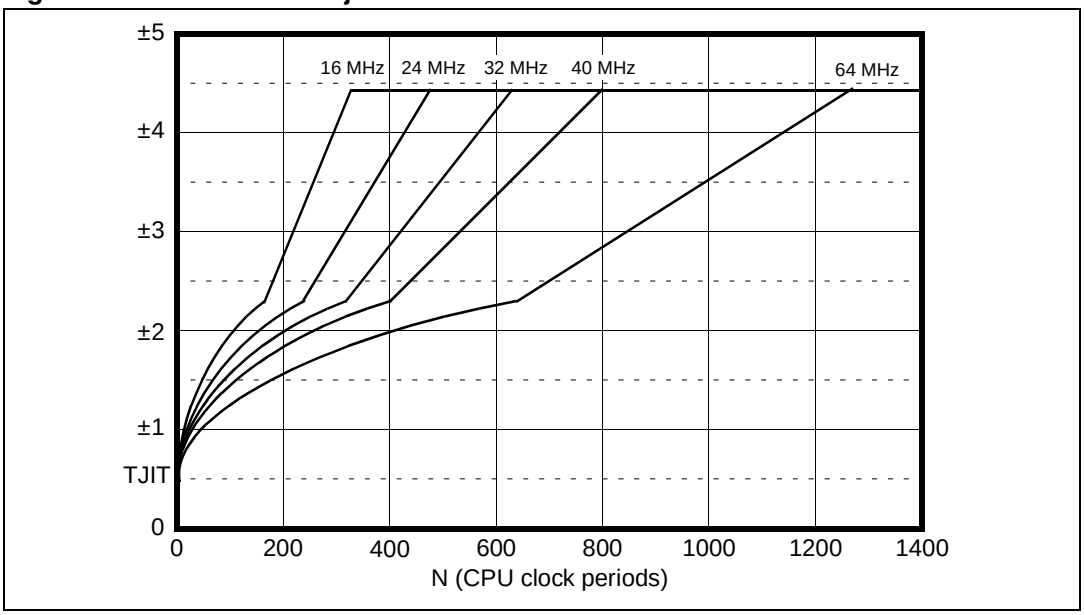
The most noise sensitive circuit in the PLL circuit is definitely the VCO (voltage controlled oscillator). There are two main sources of noise: thermal (random noise, frequency independent thus practically white noise) and flicker (low frequency noise). These two noise sources lead to a jitter as illustrated in *Figure 5*. This figure shows three distinctive zones:

- the first zone in which the R.M.S. value of the accumulated jitter is *proportional to the square root of N*, where N is the number of clock periods within the considered time interval,
- the second zone in which the R.M.S. value of the accumulated jitter is *proportional to N* and,
- the third zone where for a large value of N, a saturation effect is evident, so the jitter does not grow anymore when considering a longer time interval (jitter stable increasing the number of clock periods N). The saturation value corresponds to what has been called self referred long term jitter of the PLL. In *Figure 5* the maximum jitter trend versus the number of clock periods N (for some typical CPU frequencies) is reported: The curves represent the very worst case, computed taking into account all corners of temperature, power supply and process variations; the real jitter is always measured well below the given worst case values.

Noise in supply and substrate

Digital supply noise adds determining elements to PLL output jitter, independent of the multiplication factor. Its effect is strongly reduced thanks to particular care used in the physical implementation and integration of the PLL module inside the device. In any case, the contribution of digital noise to global jitter is widely taken into account in the curves provided in *Figure 5*.

Figure 5. ST10F27x PLL jitter



3 ST10F27x PLL jitter effect in CAN protocol

The CAN protocol provides a mechanism to resynchronize from recessive to dominant after every edge, as explained in *Section 1.3: Phase buffer segments and synchronization*. The two phase buffers and the synchronization jump make it possible to compensate the oscillator or PLL tolerance.

Considering only the PLL effects, the C-CAN module present in the ST10F27x can always compensate its PLL jitter. In the worst case, in fact, the long term jitter is +/- 4.5ns (*Figure 5 on page 16*). From the jitter point of view, one of the worst CAN bit time configurations is when $t_{BT} = 25t_q$, SJW = 1, $t_{SJW} = 1t_q = 40ns$. Considering that the summarized difference between the receiver and the transmitter is 9ns, Synchronization Jump Width is sufficient to compensate that difference. In other words, the numerator of the second term of the formula (*Section 1.4: System clock tolerance range*):

$$df < \frac{(t_{SJW}) - (2 \cdot \delta_{PLL})}{2 \cdot 10 \cdot t_{BT}}$$

is always positive, leading in any case to a system clock tolerance range. In the same way if $t_{BT} = 1\mu$, $t_{BT} = 18t_q$, $t_{PB2} = 1t_q$, the numerator of the second term of the formula:

$$df < \frac{\min(t_{PB1}, t_{PB2}) - (2 \cdot \delta_{PLL})}{(2 \cdot (13 \cdot t_{BT} - t_{PB2}))}$$

is always positive, leading again to a system clock tolerance range.

3.1 System clock tolerance range reduction in presence of PLL jitter

Even if the C-CAN module always compensates its PLL jitter, the system clock tolerance range is nevertheless reduced, increasing the probability of errors. This section evaluates that reduction and also provides two examples of configuration already dealt with in *Section 1.6: Calculation of the bit timing parameters*.

3.1.1 Range reduction percentage

$$\frac{df - df_{PLL}}{df} = \frac{2 \cdot \delta_{PLL}}{t_{SJW}}$$

$$\frac{df - df_{PLL}}{df} = \frac{2 \cdot \delta_{PLL}}{\min(t_{PB1}, t_{PB2})}$$

The above equations have been calculated starting from the relations of the system clock tolerance range. Those quantities cannot be higher than 22.5% (worst case: $\delta_{PLL} = 4.5ns$, $t_{SJW} = \min(t_{PB1}, t_{PB2}) = 40ns$).

3.1.2 Examples

In the first example of *Section 1.6*, the frequency of the CAN module clock was 20 MHz, BRP = 1 and the bit rate 1 Mbit/s. Using the *Figure 5 on page 16*, it follows that

$$\delta_{PLL}(13 \cdot 10 \cdot tq = 520 \cdot t_{CPU}) = 3ns$$

so the clock tolerance range is reduced by

$$\frac{(2 \cdot \delta_{PLL})}{(2 \cdot (13 \cdot t_{BT} - t_{PB2}))} = \frac{0.006\mu s}{2 \cdot (13 \cdot 1\mu s - 0.2\mu s)} = 0.02\%$$

becoming 0.37%.

Using the range reduction formula calculated in the previous section:

$$\frac{df - df_{PLL}}{df} = \frac{6}{100} = 0.06 = 6\%$$

In the second example, the frequency of CAN module clock was 32 MHz, BRP = 31 and the bit rate 100 Kbit/s. Using the *Figure 5 on page 16*, it follows that

$$\delta_{PLL}(13 \cdot 10 \cdot tq = 4160 \cdot t_{CPU}) = 4.5ns$$

so the clock tolerance range is reduced by

$$\frac{(2 \cdot \delta_{PLL})}{(2 \cdot (13 \cdot t_{BT} - t_{PB2}))} = \frac{0.009\mu s}{2 \cdot (13 \cdot 10\mu s - 4\mu s)} = 0.0035\%$$

which is negligible compared to the original value of 1.58%.

In fact, using the range reduction formula calculated in the previous section:

$$\frac{df - df_{PLL}}{df} = \frac{9}{4000} = 0.00225 = 0.225\%$$

3.2 Conclusion

Given the characteristics of the ST10F27x PLL, in most configurations the PLL can be used and it fulfills the CAN standard requirements.

4 Revision history

Table 2. Document revision history

Date	Revision	Changes
25-Apr-2006	1	Initial release

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZE REPRESENTATIVE OF ST, ST PRODUCTS ARE NOT DESIGNED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS, WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2006 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com